Lane Detection

TEAM MEMBERS:

1. Ujwal Kothapally
2. Venkata Krishna Sreekar Padakandla

Install Roboflow Package

In [3]:
```
!pip install roboflow
```

```
Requirement already satisfied: roboflow in /usr/local/lib/python3.10/dist-pac
kages (1.1.9)
Requirement already satisfied: certifi==2023.7.22 in /usr/local/lib/python3.1
0/dist-packages (from roboflow) (2023.7.22)
Requirement already satisfied: chardet==4.0.0 in /usr/local/lib/python3.10/di
st-packages (from roboflow) (4.0.0)
Requirement already satisfied: cycler==0.10.0 in /usr/local/lib/python3.10/di
st-packages (from roboflow) (0.10.0)
Requirement already satisfied: idna==2.10 in /usr/local/lib/python3.10/dist-p
ackages (from roboflow) (2.10)
Requirement already satisfied: kiwisolver>=1.3.1 in /usr/local/lib/python3.1
0/dist-packages (from roboflow) (1.4.5)
Requirement already satisfied: matplotlib in /usr/local/lib/python3.10/dist-p
ackages (from roboflow) (3.7.1)
Requirement already satisfied: numpy>=1.18.5 in /usr/local/lib/python3.10/dis
t-packages (from roboflow) (1.23.5)
Requirement already satisfied: opencv-python-headless==4.8.0.74 in /usr/loca
l/lib/python3.10/dist-packages (from roboflow) (4.8.0.74)
Requirement already satisfied: Pillow>=7.1.2 in /usr/local/lib/python3.10/dis
t-packages (from roboflow) (9.4.0)
Requirement already satisfied: pyparsing==2.4.7 in /usr/local/lib/python3.10/
dist-packages (from roboflow) (2.4.7)
Requirement already satisfied: python-dateutil in /usr/local/lib/python3.10/d
ist-packages (from roboflow) (2.8.2)
Requirement already satisfied: python-dotenv in /usr/local/lib/python3.10/dis
t-packages (from roboflow) (1.0.0)
Requirement already satisfied: requests in /usr/local/lib/python3.10/dist-pac
kages (from roboflow) (2.31.0)
Requirement already satisfied: six in /usr/local/lib/python3.10/dist-packages
(from roboflow) (1.16.0)
Requirement already satisfied: supervision in /usr/local/lib/python3.10/dist-
packages (from roboflow) (0.16.0)
Requirement already satisfied: urllib3>=1.26.6 in /usr/local/lib/python3.10/d
ist-packages (from roboflow) (2.0.7)
Requirement already satisfied: tqdm>=4.41.0 in /usr/local/lib/python3.10/dist
-packages (from roboflow) (4.66.1)
Requirement already satisfied: PyYAML>=5.3.1 in /usr/local/lib/python3.10/dis
t-packages (from roboflow) (6.0.1)
Requirement already satisfied: requests-toolbelt in /usr/local/lib/python3.1
0/dist-packages (from roboflow) (1.0.0)
Requirement already satisfied: python-magic in /usr/local/lib/python3.10/dist
-packages (from roboflow) (0.4.27)
Requirement already satisfied: contourpy>=1.0.1 in /usr/local/lib/python3.10/
dist-packages (from matplotlib->roboflow) (1.2.0)
Requirement already satisfied: fonttools>=4.22.0 in /usr/local/lib/python3.1
0/dist-packages (from matplotlib->roboflow) (4.44.3)
Requirement already satisfied: packaging>=20.0 in /usr/local/lib/python3.10/d
ist-packages (from matplotlib->roboflow) (23.2)
Requirement already satisfied: charset-normalizer<4,>=2 in /usr/local/lib/pyt
hon3.10/dist-packages (from requests->roboflow) (3.3.2)
Requirement already satisfied: scipy<2.0.0,>=1.9.0 in /usr/local/lib/python3.
10/dist-packages (from supervision->roboflow) (1.11.3)
```

Import Dependencies

In [4]:
```
from IPython.display import Image, clear_output
```

Install PyTorch and torchvision

In [2]:
```
!pip install torch
!pip install torchvision
```

```
Requirement already satisfied: torch in /usr/local/lib/python3.10/dist-packag
es (2.1.0+cu118)
Requirement already satisfied: filelock in /usr/local/lib/python3.10/dist-pac
kages (from torch) (3.13.1)
Requirement already satisfied: typing-extensions in /usr/local/lib/python3.1
0/dist-packages (from torch) (4.5.0)
Requirement already satisfied: sympy in /usr/local/lib/python3.10/dist-packag
es (from torch) (1.12)
Requirement already satisfied: networkx in /usr/local/lib/python3.10/dist-pac
kages (from torch) (3.2.1)
Requirement already satisfied: jinja2 in /usr/local/lib/python3.10/dist-packa
ges (from torch) (3.1.2)
Requirement already satisfied: fsspec in /usr/local/lib/python3.10/dist-packa
ges (from torch) (2023.6.0)
Requirement already satisfied: triton==2.1.0 in /usr/local/lib/python3.10/dis
t-packages (from torch) (2.1.0)
Requirement already satisfied: MarkupSafe>=2.0 in /usr/local/lib/python3.10/d
ist-packages (from jinja2->torch) (2.1.3)
Requirement already satisfied: mpmath>=0.19 in /usr/local/lib/python3.10/dist
-packages (from sympy->torch) (1.3.0)
Requirement already satisfied: torchvision in /usr/local/lib/python3.10/dist-
packages (0.16.0+cu118)
Requirement already satisfied: numpy in /usr/local/lib/python3.10/dist-packag
es (from torchvision) (1.23.5)
Requirement already satisfied: requests in /usr/local/lib/python3.10/dist-pac
kages (from torchvision) (2.31.0)
Requirement already satisfied: torch==2.1.0 in /usr/local/lib/python3.10/dist
-packages (from torchvision) (2.1.0+cu118)
Requirement already satisfied: pillow!=8.3.*,>=5.3.0 in /usr/local/lib/python
3.10/dist-packages (from torchvision) (9.4.0)
Requirement already satisfied: filelock in /usr/local/lib/python3.10/dist-pac
kages (from torch==2.1.0->torchvision) (3.13.1)
Requirement already satisfied: typing-extensions in /usr/local/lib/python3.1
0/dist-packages (from torch==2.1.0->torchvision) (4.5.0)
Requirement already satisfied: sympy in /usr/local/lib/python3.10/dist-packag
es (from torch==2.1.0->torchvision) (1.12)
Requirement already satisfied: networkx in /usr/local/lib/python3.10/dist-pac
kages (from torch==2.1.0->torchvision) (3.2.1)
Requirement already satisfied: jinja2 in /usr/local/lib/python3.10/dist-packa
ges (from torch==2.1.0->torchvision) (3.1.2)
Requirement already satisfied: fsspec in /usr/local/lib/python3.10/dist-packa
ges (from torch==2.1.0->torchvision) (2023.6.0)
Requirement already satisfied: triton==2.1.0 in /usr/local/lib/python3.10/dis
t-packages (from torch==2.1.0->torchvision) (2.1.0)
Requirement already satisfied: charset-normalizer<4,>=2 in /usr/local/lib/pyt
hon3.10/dist-packages (from requests->torchvision) (3.3.2)
Requirement already satisfied: idna<4,>=2.5 in /usr/local/lib/python3.10/dist
-packages (from requests->torchvision) (2.10)
Requirement already satisfied: urllib3<3,>=1.21.1 in /usr/local/lib/python3.1
0/dist-packages (from requests->torchvision) (2.0.7)
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.1
0/dist-packages (from requests->torchvision) (2023.7.22)
Requirement already satisfied: MarkupSafe>=2.0 in /usr/local/lib/python3.10/d
ist-packages (from jinja2->torch==2.1.0->torchvision) (2.1.3)
Requirement already satisfied: mpmath>=0.19 in /usr/local/lib/python3.10/dist
-packages (from sympy->torch==2.1.0->torchvision) (1.3.0)
```

In [5]:
```
!pip install --upgrade pip
```

```
Requirement already satisfied: pip in /usr/local/lib/python3.10/dist-packages
(23.1.2)
Collecting pip
  Downloading pip-23.3.1-py3-none-any.whl (2.1 MB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 2.1/2.1 MB 12.5 MB/s eta 0:00:0
0
Installing collected packages: pip
  Attempting uninstall: pip
    Found existing installation: pip 23.1.2
    Uninstalling pip-23.1.2:
      Successfully uninstalled pip-23.1.2
Successfully installed pip-23.3.1
```

Import PyTorch and Other Libraries

In [6]:
```
from roboflow import Roboflow
import torch
from torch.utils.data import Dataset, DataLoader
from torchvision import transforms
from torchvision.datasets import ImageFolder
from sklearn.model_selection import train_test_split
import matplotlib.pyplot as plt
import torch.nn as nn
import torchvision.models as models
from torchvision import transforms
import torch.optim as optim
```

Downloads a dataset from Roboflow using the YOLOv5 model format, specifying the API key, project name and version, and saves it

In [7]:
```
rf = Roboflow(api_key="8XxgFmL7Zjg4BCle3kc4", model_format="yolov5")
dataset = rf.workspace().project("lane_detection-rd6mu").version(1).download(l
```

```
loading Roboflow workspace...
loading Roboflow project...

Downloading Dataset Version Zip in /Downloads to yolov5pytorch:: 100%|███████
███| 13630/13630 [00:00<00:00, 38871.19it/s]



Extracting Dataset Version Zip to /Downloads in yolov5pytorch:: 100%|████████
██| 320/320 [00:00<00:00, 6476.31it/s]
```

Custom PyTorch dataset class (CustomDataset) that wraps around the ImageFolder class, allowing for **customization of the data loading process, sets a seed for reproducibility, defines image transformations

In [8]:
```
class CustomDataset(Dataset):
    def __init__(self, root_dir, transform=None):
        self.dataset = ImageFolder(root_dir, transform=transform)

    def __len__(self):
        return len(self.dataset)

    def __getitem__(self, idx):
        return self.dataset[idx]

# Set seed for reproducibility
torch.manual_seed(42)

# Define transforms
transform = transforms.Compose([
    transforms.Resize((224, 224)),
    transforms.ToTensor(),
])

# Load dataset
dataset = CustomDataset(root_dir="/Downloads", transform=transform)
```

Custom PyTorch neural network model (LaneDetectionModel) based on the DeepLabV3 architecture with a ResNet-101 backbone, loads pre-trained weights, and modifies the output layer to suit the task of lane detection by changing the number of output classes.

In [9]:
```
class LaneDetectionModel(nn.Module):
    def __init__(self, num_classes=1):
        super(LaneDetectionModel, self).__init__()

        # Load pre-trained DeepLabV3 model
        self.deeplabv3 = models.segmentation.deeplabv3_resnet101(pretrained=Tr

        # Modify the output layer based on your needs
        in_channels = self.deeplabv3.classifier[-1].in_channels
        self.deeplabv3.classifier[-1] = nn.Conv2d(in_channels, num_classes, ke

    def forward(self, x):
        return self.deeplabv3(x)['out']

# Create an instance of the model
your_model = LaneDetectionModel()

# Print the model architecture
print(your_model)
```

```
-586e9e4e.pth" to /root/.cache/torch/hub/checkpoints/deeplabv3_resnet101_c
oco-586e9e4e.pth
100%|██████████| 233M/233M [00:02<00:00, 83.9MB/s]

LaneDetectionModel(
  (deeplabv3): DeepLabV3(
    (backbone): IntermediateLayerGetter(
      (conv1): Conv2d(3, 64, kernel_size=(7, 7), stride=(2, 2), padding=
(3, 3), bias=False)
      (bn1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_r
unning_stats=True)
      (relu): ReLU(inplace=True)
      (maxpool): MaxPool2d(kernel_size=3, stride=2, padding=1, dilation=1,
ceil_mode=False)
      (layer1): Sequential(
        (0): Bottleneck(
          (conv1): Conv2d(64, 64, kernel_size=(1, 1), stride=(1, 1), bias=
False)
          (bn1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, tra
ck_running_stats=True)
```

Instantiates an instance of the previously defined LaneDetectionModel with one output class, sets up Mean Squared Error (MSE) loss, defines an Adam optimizer with a learning rate of 0.001, specifies the number of training epochs (5), and splits the dataset into training, validation, and test sets using random splitting. Finally, creates DataLoader instances for the training, validation, and test sets with batch size 8

In [10]:
```
your_model = LaneDetectionModel(num_classes=1)
criterion = nn.MSELoss()
# Define optimizer (e.g., Adam)
optimizer = optim.Adam(your_model.parameters(), lr=0.001)
num_epochs = 5
# Split the dataset into train, validation, and test sets
train_size = int(0.8 * len(dataset))
val_size = int(0.1 * len(dataset))
test_size = len(dataset) - train_size - val_size

train_dataset, temp_dataset = torch.utils.data.random_split(dataset, [train_si
val_dataset, test_dataset = torch.utils.data.random_split(temp_dataset, [val_s

# Create DataLoader
train_loader = DataLoader(train_dataset, batch_size=8, shuffle=True)
val_loader = DataLoader(val_dataset, batch_size=8, shuffle=False)
test_loader = DataLoader(test_dataset, batch_size=8, shuffle=False)
```

Set of image transforms, including resizing, random horizontal flipping, color jittering, converting to a PyTorch tensor, and normalization.

In [11]:
```
# Define transforms for normalization and augmentation
transform = transforms.Compose([
    transforms.Resize((224, 224)),
    transforms.RandomHorizontalFlip(),
    transforms.ColorJitter(brightness=0.2, contrast=0.2, saturation=0.2, hue=0
    transforms.ToTensor(),
    transforms.Normalize(mean=[0.485, 0.456, 0.406], std=[0.229, 0.224, 0.225]
])

# Load dataset with the defined transforms
dataset = CustomDataset(root_dir="/Downloads", transform=transform)
```

A function (visualize_samples) to visualize the first sample of each minibatch from a given DataLoader, creates a DataLoader for visualization with a batch size of 8 and shuffling

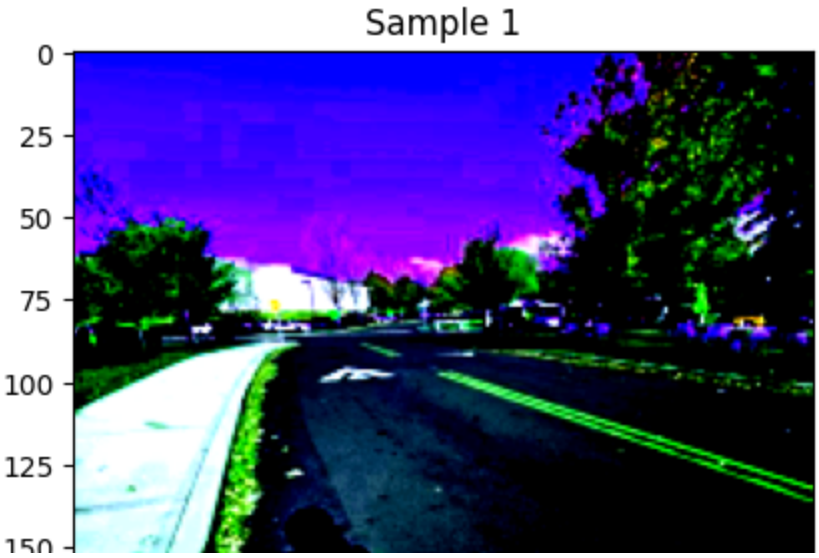
In [12]:
```
# Visualize the 1st sample of each minibatch of size 8
def visualize_samples(loader):
    data_iter = iter(loader)
    images, _ = next(data_iter)

    for i in range(images.shape[0]):
        plt.figure()
        plt.imshow(images[i].permute(1, 2, 0))
        plt.title(f"Sample {i+1}")
        plt.show()

# Create DataLoader for visualization
visualize_loader = DataLoader(dataset, batch_size=8, shuffle=True)

# Visualize the 1st sample of each minibatch
visualize_samples(visualize_loader)
```

```
WARNING:matplotlib.image:Clipping input data to the valid range for imshow
with RGB data ([0..1] for floats or [0..255] for integers).
```



Set the neural network model (your_model) to training mode, iterates through epochs and mini-batches in the training DataLoader (train_loader), performs a forward pass, calculates the Mean Squared Error (MSE) loss between the model's output and the target, backpropagates the gradients, and updates the model's weights using the Adam optimizer.

In [13]:
```
your_model.train()  # Set the model to training mode
#target = target.view(-1, 1, 1, 1).expand_as(output)
for epoch in range(num_epochs):
    for batch_idx, (data, target) in enumerate(train_loader):
        if batch_idx == 0:  # Only for the first mini-batch
            optimizer.zero_grad()  # Zero the gradients
            output = your_model(data)  # Forward pass
            target = target.view(-1, 1, 1, 1).expand_as(output).float()
            print(f"Model Output Size: {output.shape}")
            loss = criterion(output, target)  # Calculate the loss
            loss.backward()  # Backward pass
            optimizer.step()  # Update the weights
            print(f"Epoch {epoch+1}, Batch {batch_idx+1}, Loss: {loss.item()}"
```

```
Model Output Size: torch.Size([8, 1, 224, 224])
Epoch 1, Batch 1, Loss: 1.4757877588272095
Model Output Size: torch.Size([8, 1, 224, 224])
Epoch 2, Batch 1, Loss: 2.2448413372039795
Model Output Size: torch.Size([8, 1, 224, 224])
Epoch 3, Batch 1, Loss: 1.8363642692565918
Model Output Size: torch.Size([8, 1, 224, 224])
Epoch 4, Batch 1, Loss: 1.3391704559326172
Model Output Size: torch.Size([8, 1, 224, 224])
Epoch 5, Batch 1, Loss: 0.8838760852813721
```

Transform in the CustomDataset class, setting it to resize images to 32 by 32 pixels, convert them to PyTorch tensors, and normalize the pixel values

In [14]:
```
transform = transforms.Compose([
    transforms.Resize((32, 32)),  # Resize to 32 by 32
    transforms.ToTensor(),
    transforms.Normalize(mean=[0.485, 0.456, 0.406], std=[0.229, 0.224, 0.225]
])

# Update the transform in CustomDataset
dataset = CustomDataset(root_dir="/Downloads", transform=transform)
```

Update the optimizer with a new learning rate, perform a validation loop to find a suitable learning rate by evaluating the model on the validation set, and then updates the optimizer with the chosen learning rate. Finally, trains the model using the updated optimizer on the training set for the specified number of epochs. The training progress, including epoch number,

batch number, and loss, is printed during training.

In [15]:
```
from torch.optim import lr_scheduler
learning_rates = [0.01, 0.1]
best_val_loss = float('inf')  # Initialize with a large value

for lr in learning_rates:
    optimizer = optim.Adam(your_model.parameters(), lr=lr)
    scheduler = lr_scheduler.StepLR(optimizer, step_size=1, gamma=0.1)

    for epoch in range(num_epochs):
        your_model.train()
        train_loss = 0.0
        for batch_idx, (data, target) in enumerate(train_loader):
            optimizer.zero_grad()
            output = your_model(data)
            target = target.view(-1, 1, 1, 1).expand_as(output).float()
            loss = criterion(output, target)
            loss.backward()
            optimizer.step()
            train_loss += loss.item()

        scheduler.step()

        # Validation loop
        your_model.eval()
        val_loss = 0.0
        for batch_idx, (data, target) in enumerate(val_loader):
            with torch.no_grad():
                output = your_model(data)
                target = target.view(-1, 1, 1, 1).expand_as(output).float()
                loss = criterion(output, target)
                val_loss += loss.item()

        avg_train_loss = train_loss / len(train_loader)
        avg_val_loss = val_loss / len(val_loader)
        print(f"Epoch {epoch+1}, Learning rate: {optimizer.param_groups[0]['lr
              f"Train Loss: {avg_train_loss}, Validation Loss: {avg_val_loss}"

        # Save the model if it has the best validation loss
        if avg_val_loss < best_val_loss:
            best_val_loss = avg_val_loss
            torch.save(your_model.state_dict(), 'best_model.pth')
```

```
Epoch 1, Learning rate: 0.001, Train Loss: 0.656825833953917, Validation Los
s: 1.1453181411826074e+18
Epoch 2, Learning rate: 0.0001, Train Loss: 0.29108639247715473, Validation L
oss: 1018.5716247558594
Epoch 3, Learning rate: 1e-05, Train Loss: 0.25507048168219626, Validation Lo
ss: 4.690543591976166
Epoch 4, Learning rate: 1.0000000000000002e-06, Train Loss: 0.254588124342262
74, Validation Loss: 1.9085792005062103
Epoch 5, Learning rate: 1.0000000000000002e-07, Train Loss: 0.260748941916972
4, Validation Loss: 1.0588766634464264
Epoch 1, Learning rate: 0.010000000000000002, Train Loss: 4.955032827332616,
Validation Loss: 8.15484778058606e+27
Epoch 2, Learning rate: 0.0010000000000000002, Train Loss: 0.356736313784495,
Validation Loss: 13388.47802734375
Epoch 3, Learning rate: 0.00010000000000000003, Train Loss: 0.308627081103622
9, Validation Loss: 11.351924180984497
Epoch 4, Learning rate: 1.0000000000000004e-05, Train Loss: 0.287053593667224
05, Validation Loss: 1.1289360523223877
Epoch 5, Learning rate: 1.0000000000000004e-06, Train Loss: 0.286416980903595
7, Validation Loss: 0.8454495668411255
```

In [16]:
```
# After finding a suitable learning rate, update the optimizer
optimizer = optim.Adam(your_model.parameters(), lr=0.001)  # Update with the c

# Train the model with the updated optimizer
your_model.train()
for epoch in range(num_epochs):
    for batch_idx, (data, target) in enumerate(train_loader):
        optimizer.zero_grad()
        output = your_model(data)
        target = target.view(-1, 1, 1, 1).expand_as(output).float()
        loss = criterion(output, target)
        loss.backward()
        optimizer.step()
        print(f"Epoch {epoch+1}, Batch {batch_idx+1}, Loss: {loss.item()}")
```

```
Epoch 1, Batch 1, Loss: 0.41538751125335693
Epoch 1, Batch 2, Loss: 0.12446936964988708
Epoch 1, Batch 3, Loss: 0.3777618706226349
Epoch 1, Batch 4, Loss: 0.12082774192094803
Epoch 1, Batch 5, Loss: 0.37840521335601807
Epoch 1, Batch 6, Loss: 0.01618461310863495
Epoch 1, Batch 7, Loss: 0.410344660282135
Epoch 1, Batch 8, Loss: 0.13516566157341003
Epoch 1, Batch 9, Loss: 0.013834677636623383
Epoch 1, Batch 10, Loss: 0.11406020820140839
Epoch 1, Batch 11, Loss: 0.5083169937133789
Epoch 1, Batch 12, Loss: 0.2642454504966736
Epoch 1, Batch 13, Loss: 0.26378366351127625
Epoch 1, Batch 14, Loss: 0.321753591299057
Epoch 1, Batch 15, Loss: 0.4876828193664551
Epoch 1, Batch 16, Loss: 0.2726232409477234
Epoch 2, Batch 1, Loss: 0.018230829387903214
Epoch 2, Batch 2, Loss: 0.21417446434497833
Epoch 2, Batch 3, Loss: 0.2984607219696045
Epoch 2, Batch 4, Loss: 0.5025426745414734
Epoch 2, Batch 5, Loss: 0.19087277352809906
Epoch 2, Batch 6, Loss: 0.1225915253162384
Epoch 2, Batch 7, Loss: 0.030401967465877533
Epoch 2, Batch 8, Loss: 0.27373385429382324
Epoch 2, Batch 9, Loss: 0.395318865776062
Epoch 2, Batch 10, Loss: 0.2776070535182953
Epoch 2, Batch 11, Loss: 0.3117884695529938
Epoch 2, Batch 12, Loss: 0.11949557065963745
Epoch 2, Batch 13, Loss: 0.5357749462127686
Epoch 2, Batch 14, Loss: 0.5027633309364319
Epoch 2, Batch 15, Loss: 0.11765647679567337
Epoch 2, Batch 16, Loss: 0.2567930519580841
Epoch 3, Batch 1, Loss: 0.31313440203666687
Epoch 3, Batch 2, Loss: 0.21236510574817657
Epoch 3, Batch 3, Loss: 0.4422813653945923
Epoch 3, Batch 4, Loss: 0.20786012709140778
Epoch 3, Batch 5, Loss: 0.20281371474266052
Epoch 3, Batch 6, Loss: 0.35194075107574463
Epoch 3, Batch 7, Loss: 0.19646111130714417
Epoch 3, Batch 8, Loss: 0.2797708213329315
Epoch 3, Batch 9, Loss: 0.4445939064025879
Epoch 3, Batch 10, Loss: 0.34571734070777893
Epoch 3, Batch 11, Loss: 0.20331302285194397
Epoch 3, Batch 12, Loss: 0.227029487490654
Epoch 3, Batch 13, Loss: 0.2071797251701355
Epoch 3, Batch 14, Loss: 0.2965186834335327
Epoch 3, Batch 15, Loss: 0.11168753355741501
Epoch 3, Batch 16, Loss: 0.015487655065953732
Epoch 4, Batch 1, Loss: 0.21220284700393677
Epoch 4, Batch 2, Loss: 0.2746049463748932
Epoch 4, Batch 3, Loss: 0.11400426179170609
Epoch 4, Batch 4, Loss: 0.4678858518600464
Epoch 4, Batch 5, Loss: 0.4073689579963684
Epoch 4, Batch 6, Loss: 0.14390650391578674
Epoch 4, Batch 7, Loss: 0.01139782927930355
Epoch 4, Batch 8, Loss: 0.01025058701634407
Epoch 4, Batch 9, Loss: 0.2996542453765869
```

```
Epoch 4, Batch 10, Loss: 0.5348249077796936
Epoch 4, Batch 11, Loss: 0.3477481007575989
Epoch 4, Batch 12, Loss: 0.2514326572418213
Epoch 4, Batch 13, Loss: 0.12726934254169464
Epoch 4, Batch 14, Loss: 0.35609856247901917
Epoch 4, Batch 15, Loss: 0.3520612418651581
Epoch 4, Batch 16, Loss: 0.3095180094242096
Epoch 5, Batch 1, Loss: 0.25597116351127625
Epoch 5, Batch 2, Loss: 0.25895363092422485
Epoch 5, Batch 3, Loss: 0.42498674988746643
Epoch 5, Batch 4, Loss: 0.4756758213043213
Epoch 5, Batch 5, Loss: 0.01614869385957718
Epoch 5, Batch 6, Loss: 0.16965314745903015
Epoch 5, Batch 7, Loss: 0.11729030311107635
Epoch 5, Batch 8, Loss: 0.1775282621383667
Epoch 5, Batch 9, Loss: 0.11302187293767929
Epoch 5, Batch 10, Loss: 0.11521869152784348
Epoch 5, Batch 11, Loss: 0.1781185120344162
Epoch 5, Batch 12, Loss: 0.11200012266635895
Epoch 5, Batch 13, Loss: 0.6111540198326111
Epoch 5, Batch 14, Loss: 0.5228303074836731
Epoch 5, Batch 15, Loss: 0.3103388845920563
Epoch 5, Batch 16, Loss: 0.26817944645881653
```

Evaluate the trained model on the test set in evaluation mode, calculating the Mean Squared Error (MSE) and Mean Absolute Error (MAE) for each mini-batch, and then computes the average MSE and MAE over the entire test set.

In [17]:
```
your_model.eval()  # Set the model to evaluation mode
total_mse = 0.0
total_mae = 0.0
num_samples = 0

for data, target in test_loader:
    with torch.no_grad():
        output = your_model(data)
        target = target.view(-1, 1, 1, 1).expand_as(output).float()

        # Calculate Mean Squared Error (MSE)
        mse = nn.MSELoss()(output, target)
        total_mse += mse.item()

        # Calculate Mean Absolute Error (MAE)
        mae = nn.L1Loss()(output, target)
        total_mae += mae.item()

        num_samples += data.size(0)

# Calculate average metrics
avg_mse = total_mse / len(test_loader)
avg_mae = total_mae / len(test_loader)
```

In [18]:
```
print("Average MSE:",avg_mse)
print("Average MAE:",avg_mae)
```

```
Average MSE: 0.40342994034290314
Average MAE: 0.47346214950084686
```

In [19]:
```
import torch
from sklearn.metrics import f1_score, accuracy_score
import numpy as np

your_model.eval()  # Set the model to evaluation mode
threshold = 0.5

true_labels = []
predicted_labels = []

for data, target in test_loader:
    with torch.no_grad():
        output = your_model(data)
        target = target.view(-1, 1, 1, 1).expand_as(output).float()

        # Convert continuous predictions to binary
        binary_predictions = (output > threshold).float()

        true_labels.extend(target.cpu().numpy().flatten())
        predicted_labels.extend(binary_predictions.cpu().numpy().flatten())

# Convert lists to numpy arrays
true_labels = np.array(true_labels)
predicted_labels = np.array(predicted_labels)

# Convert to binary labels based on the threshold
binary_true_labels = (true_labels > threshold).astype(int)
binary_predicted_labels = (predicted_labels > threshold).astype(int)

# Calculate F1 score
f1 = f1_score(binary_true_labels, binary_predicted_labels)

# Calculate Accuracy
accuracy = accuracy_score(binary_true_labels, binary_predicted_labels)

print(f"F1 Score: {f1:.4f}")
print(f"Accuracy: {accuracy:.4f}")
```

```
F1 Score: 0.9333
Accuracy: 0.8750
```